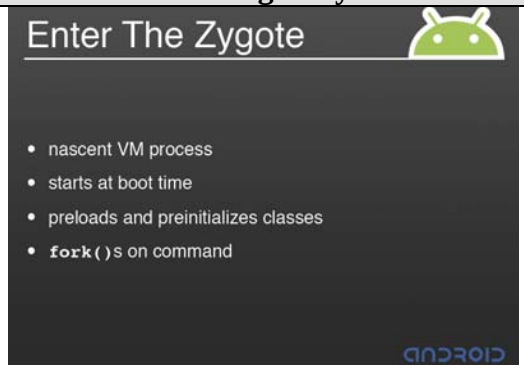
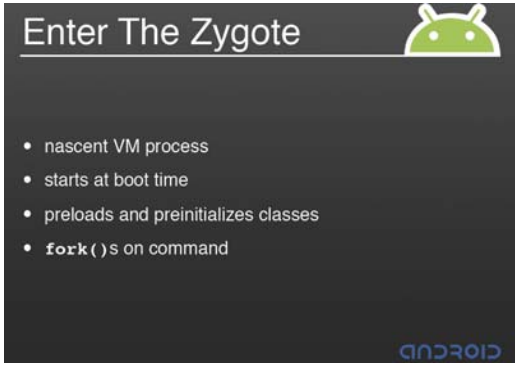
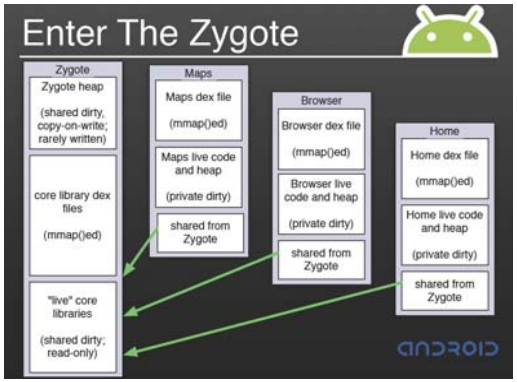
Exhibit C

The '720 Patent	Infringed By
	 (Dalvik Presentation, Slide 25) Corresponding Dalvik Video at 13:48: "What we do with the zygote, as its name implies,…when it gets a command to start up a new application, it does a normal Unix fork and then that child process becomes that target application. And the result of that is this." *See also* claim 1.f. below.
1.f. a copy-on-write process cloning mechanism to instantiate the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process, and to defer copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the master runtime system process.	Android includes a copy-on-write process cloning mechanism to instantiate the child runtime system process by copying references to the memory space of the master runtime system process into a separate memory space for the child runtime system process, and to defer copying of the memory space of the master runtime system process until the child runtime system process needs to modify the referenced memory space of the master runtime system process. *See*

The '720 Patent	Infringed By
	 (Android Presentation, Slide 82) Corresponding Android Video at 44:30: "The init process starts up a really neat process called zygote….It uses copy-on-write to maximize re-use and minimize footprint so that data structures are shared and it won't do a full copy unless some of those data structures are to be modified." *See also*  (Dalvik Presentation, Slide 25) Corresponding Dalvik Video at 13:48:

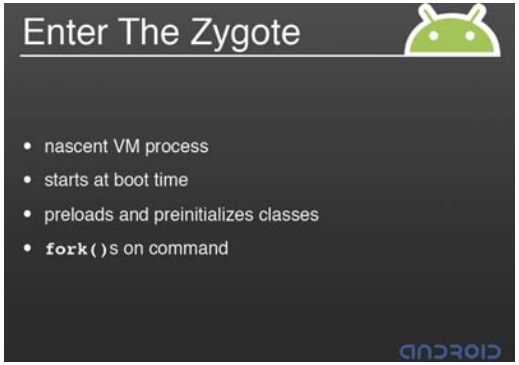
The '720 Patent	Infringed By
	"What we do with the zygote, as its name implies,…when it gets a command to start up a new application, it does a normal Unix fork and then that child process becomes that target application. And the result of that is this."  (Dalvik Presentation, Slide 26) Corresponding Dalvik Video at 14:40: "So the zygote, again, has made, has made this heap of objects, it's made this live dex structure and then each application that then starts up, instead of having its own memory for those things, it just shares it with the zygote and also with any other app that's also on the system." *See also* http://developer.android.com/guide/basics/what-is-android.html. "Android Runtime …The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management. Linux Kernel Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as

The '720 Patent	Infringed By
	an abstraction layer between the hardware and the rest of the software stack." *See also*, Lowe, Robert, Linux Kernel Process Management, April 15, 2005. Sample Chapter is provided courtesy of Sams, http://www.informit.com/articles/article.aspx?p=370047&seqNum=2&rll=1. "Copy-on-Write …In Linux, fork() is implemented through the use of copy-on-write pages. Copy-on-write (or COW) is a technique to delay or altogether prevent copying of the data. Rather than duplicate the process address space, the parent and the child can share a single copy. The data, however, is marked in such a way that if it is written to, a duplicate is made and each process receives a unique copy." Example source code files in libcore\dalvik\src\main\java\dalvik\system\Zygote.java, dalvik\vm\native\dalvik_system_Zygote.c, linux-2.6\kernel\fork.c. Example code call chain forkAndSpecialize calls forkAndSpecializeCommon, forkAndSpecializeCommon calls fork, Linux fork process do_fork calls copy_process. *See, e.g.*, libcore\dalvik\src\main\java\dalvik\system\Zygote.java. */*** ** Forks a new Zygote instance, but does not leave the zygote mode.* ** The current VM must have been started with the -Xzygote flag. The* ** new child is expected to eventually call forkAndSpecialize()* *** ** @return 0 if this is the child, pid of the child*

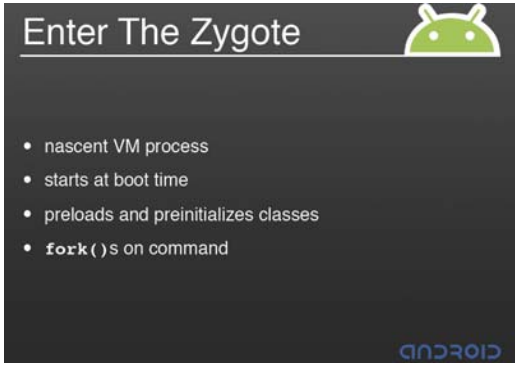
The '720 Patent	Infringed By
	** if this is the parent, or -1 on error* **/* *native public static int fork();* */*** ** Forks a new VM instance. The current VM must have been started* ** with the -Xzygote flag. NOTE: new instance keeps all* ** root capabilities. The new process is expected to call capset().* *** ** @param uid the UNIX uid that the new process should setuid() to after* ** fork()ing and and before spawning any threads.* ** @param gid the UNIX gid that the new process should setgid() to after* ** fork()ing and and before spawning any threads.* ** @param gids null-ok; a list of UNIX gids that the new process should* ** setgroups() to after fork and before spawning any threads.* ** @param debugFlags bit flags that enable debugging features.* ** @param rlimits null-ok an array of rlimit tuples, with the second* ** dimension having a length of 3 and representing* ** (resource, rlim_cur, rlim_max). These are set via the posix* ** setrlimit(2) call.* *** ** @return 0 if this is the child, pid of the child* ** if this is the parent, or -1 on error.* **/* *native public static int forkAndSpecialize(int uid, int gid, int[] gids,* *int debugFlags, int[][] rlimits);* *See, e.g.*, dalvik\vm\native\dalvik_system_Zygote.c. */* native public static int forkAndSpecialize(int uid, int gid,* ** int[] gids, int debugFlags);* **/* *static void Dalvik_dalvik_system_Zygote_forkAndSpecialize(const u4* args,* *JValue* pResult)* *{* *pid_t pid;* *pid = forkAndSpecializeCommon(args);* *RETURN_INT(pid);*

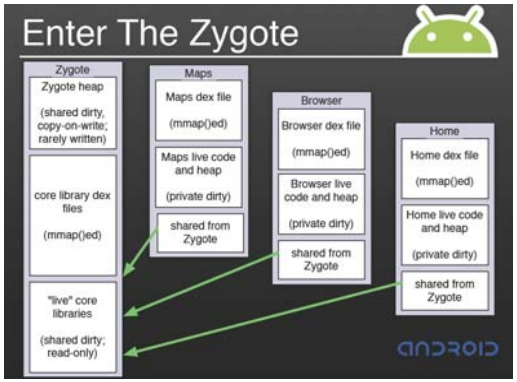
The ’720 Patent	Infringed By
	} *…* */* ** Utility routine to fork zygote and specialize the child process.* **/* *static pid_t forkAndSpecializeCommon(const u4* args, bool isSystemServer)* *{* *pid_t pid;* *uid_t uid = (uid_t) args[0];* *gid_t gid = (gid_t) args[1];* *ArrayObject* gids = (ArrayObject *)args[2];* *u4 debugFlags = args[3];* *ArrayObject *rlimits = (ArrayObject *)args[4];* *int64_t permittedCapabilities, effectiveCapabilities;* *if (isSystemServer) {* */** ** Don't use GET_ARG_LONG here for now. gcc is generating code* ** that uses register d8 as a temporary, and that's coming out* ** scrambled in the child process. b/3138621* **/* *//permittedCapabilities = GET_ARG_LONG(args, 5);* *//effectiveCapabilities = GET_ARG_LONG(args, 7);* *permittedCapabilities = args[5] | (int64_t) args[6] << 32;* *effectiveCapabilities = args[7] | (int64_t) args[8] << 32;* *} else {* *permittedCapabilities = effectiveCapabilities = 0;* *}* *if (!gDvm.zygote) {* *dvmThrowException("Ljava/lang/IllegalStateException;",* *"VM instance not started with -Xzygote");* *return -1;* *}* *if (!dvmGcPreZygoteFork()) {* *LOGE("pre-fork heap failed\n");*

The '720 Patent	Infringed By
	dvmAbort(); *}* *setSignalHandler();* *dvmDumpLoaderStats("zygote");* *pid = fork();* *if (pid == 0) {* *int err;* */* The child process */* …. *} else if (pid > 0) {* */* the parent process */* *}* *return pid;* *}* *See, e.g.*, linux-2.6\kernel\fork.c. */** ** Ok, this is the main fork-routine.* *** ** It copies the process, and if successful kick-starts* ** it and waits for it to finish using the VM if required.* **/* *long do_fork(unsigned long clone_flags,* *unsigned long stack_start,* *struct pt_regs *regs,* *unsigned long stack_size,* *int __user *parent_tidptr,* *int __user *child_tidptr)* *{* *struct task_struct *p;* *int trace = 0;* *long nr;* … *p = copy_process(clone_flags, stack_start, regs, stack_size,*

The '720 Patent	Infringed By
	wake_up_new_task(p, clone_flags); … *tracehook_report_clone_complete(trace, regs, clone_flags, nr, p);* … *return nr;* *}*
2. A system according to claim 1, further comprising: a cache checker to determine whether the instantiated class definition is available in a local cache associated with the master runtime system process.	Android includes a cache checker to determine whether the instantiated class definition is available in a local cache associated with the master runtime system process. *See*  (Dalvik Presentation, Slide 25) Corresponding Dalvik Video at 13:48: "What we do with the zygote, as its name implies, it's, it comes into existence fairly early on during the boot of an Android system and its job is to load up those classes that we believe will be used across many applications. So it goes and creates, it goes and creates a heap, it goes and creates that dirty memory for all, to represent those classes and methods…." Example source code files in dalvik\vm\oo\Class.c, dalvik\vm\native\java_lang_Class.c,

The '720 Patent	Infringed By
	struct stat fdStat, fileStat; *bool readOnly = false;* **pNewFile = false;* *retry:* */** ** Try to open the cache file. If we've been asked to,* ** create it if it doesn't exist.* **/* *fd = createIfMissing ? open(cacheFileName, O_CREAT|O_RDWR, 0644) : -1;* *if (fd < 0) {* *fd = open(cacheFileName, O_RDONLY, 0);* *if (fd < 0) {* *if (createIfMissing) {* *LOGE("Can't open dex cache '%s': %s\n",* *cacheFileName, strerror(errno));* *}* *return fd;* *}* *readOnly = true;* *}* … *}*
6. A system according to claim 1, further comprising: a process cloning mechanism to instantiate the child runtime system process by copying the memory space of the master runtime system process into a separate memory space for the child runtime system process.	Android includes a process cloning mechanism to instantiate a child runtime system process by copying the memory space of a master runtime system process into a separate memory space for the child runtime system process. *See*

The '720 Patent	Infringed By
	 (Android Presentation, Slide 82) Corresponding Android Video at 44:30: "The init process starts up a really neat process called zygote….It uses copy-on-write to maximize re-use and minimize footprint so that data structures are shared and it won't do a full copy unless some of those data structures are to be modified." *See also*  (Dalvik Presentation, Slide 25) Corresponding Dalvik Video at 13:48:

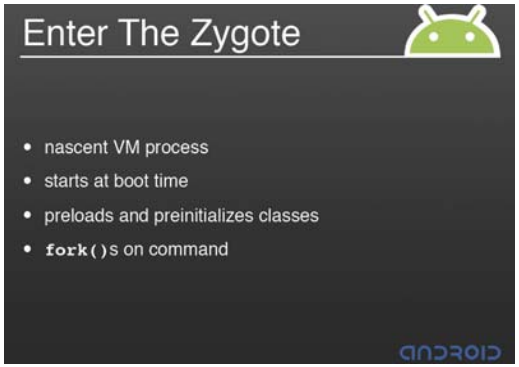
The '720 Patent	Infringed By
	"What we do with the zygote, as its name implies,…when it gets a command to start up a new application, it does a normal Unix fork and then that child process becomes that target application. And the result of that is this."  (Dalvik Presentation, Slide 26) Corresponding Dalvik Video at 14:40: "So the zygote, again, has made, has made this heap of objects, it's made this live dex structure and then each application that then starts up, instead of having its own memory for those things, it just shares it with the zygote and also with any other app that's also on the system." *See also* http://developer.android.com/guide/basics/what-is-android.html. "Android Runtime …The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management. Linux Kernel Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as

The '720 Patent	Infringed By
	an abstraction layer between the hardware and the rest of the software stack." *See also*, Lowe, Robert, Linux Kernel Process Management, April 15, 2005. Sample Chapter is provided courtesy of Sams, http://www.informit.com/articles/article.aspx?p=370047&seqNum=2&rll=1. "Copy-on-Write …In Linux, fork() is implemented through the use of copy-on-write pages. Copy-on-write (or COW) is a technique to delay or altogether prevent copying of the data. Rather than duplicate the process address space, the parent and the child can share a single copy. The data, however, is marked in such a way that if it is written to, a duplicate is made and each process receives a unique copy." Example source code files in libcore\dalvik\src\main\java\dalvik\system\Zygote.java, dalvik\vm\native\dalvik_system_Zygote.c, linux-2.6\kernel\fork.c, external\kernel-headers\original\linux\sched.h. *See, e.g.*, libcore\dalvik\src\main\java\dalvik\system\Zygote.java. */*** ** Forks a new Zygote instance, but does not leave the zygote mode.* ** The current VM must have been started with the -Xzygote flag. The* ** new child is expected to eventually call forkAndSpecialize()* *** ** @return 0 if this is the child, pid of the child* ** if this is the parent, or -1 on error* **/* *native public static int fork();* */***

The '720 Patent	Infringed By
	** Forks a new VM instance. The current VM must have been started* ** with the -Xzygote flag. NOTE: new instance keeps all* ** root capabilities. The new process is expected to call capset().* *** ** @param uid the UNIX uid that the new process should setuid() to after* ** fork()ing and and before spawning any threads.* ** @param gid the UNIX gid that the new process should setgid() to after* ** fork()ing and and before spawning any threads.* ** @param gids null-ok; a list of UNIX gids that the new process should* ** setgroups() to after fork and before spawning any threads.* ** @param debugFlags bit flags that enable debugging features.* ** @param rlimits null-ok an array of rlimit tuples, with the second* ** dimension having a length of 3 and representing* ** (resource, rlim_cur, rlim_max). These are set via the posix* ** setrlimit(2) call.* *** ** @return 0 if this is the child, pid of the child* ** if this is the parent, or -1 on error.* **/* *native public static int forkAndSpecialize(int uid, int gid, int[] gids,* *int debugFlags, int[][] rlimits);* *See, e.g.*, dalvik\vm\native\dalvik_system_Zygote.c. */* native public static int forkAndSpecialize(int uid, int gid,* ** int[] gids, int debugFlags);* **/* *static void Dalvik_dalvik_system_Zygote_forkAndSpecialize(const u4* args, JValue* pResult)* *{* *pid_t pid;* *pid = forkAndSpecializeCommon(args);* *RETURN_INT(pid);* *}* … */** ** Utility routine to fork zygote and specialize the child process.* **/*

The '720 Patent	Infringed By
	static pid_t forkAndSpecializeCommon(const u4 args, bool isSystemServer)* *{* *pid_t pid;* *uid_t uid = (uid_t) args[0];* *gid_t gid = (gid_t) args[1];* *ArrayObject* gids = (ArrayObject *)args[2];* *u4 debugFlags = args[3];* *ArrayObject *rlimits = (ArrayObject *)args[4];* *int64_t permittedCapabilities, effectiveCapabilities;* *if (isSystemServer) {* */* ** Don't use GET_ARG_LONG here for now. gcc is generating code* ** that uses register d8 as a temporary, and that's coming out* ** scrambled in the child process. b/3138621* **/* *//permittedCapabilities = GET_ARG_LONG(args, 5);* *//effectiveCapabilities = GET_ARG_LONG(args, 7);* *permittedCapabilities = args[5] | (int64_t) args[6] << 32;* *effectiveCapabilities = args[7] | (int64_t) args[8] << 32;* *} else {* *permittedCapabilities = effectiveCapabilities = 0;* *}* *if (!gDvm.zygote) {* *dvmThrowException("Ljava/lang/IllegalStateException;",* *"VM instance not started with -Xzygote");* *return -1;* *}* *if (!dvmGcPreZygoteFork()) {* *LOGE("pre-fork heap failed\n");* *dvmAbort();* *}* *setSignalHandler();*

The '720 Patent	Infringed By
	dvmDumpLoaderStats("zygote"); *pid = fork();* *if (pid == 0) {* *int err;* */* The child process */* …. *} else if (pid > 0) {* */* the parent process */* *}* *return pid;* *}* *See, e.g.*, linux-2.6\kernel\fork.c. */* ** Ok, this is the main fork-routine.* *** ** It copies the process, and if successful kick-starts* ** it and waits for it to finish using the VM if required.* **/* *long do_fork(unsigned long clone_flags,* *unsigned long stack_start,* *struct pt_regs *regs,* *unsigned long stack_size,* *int __user *parent_tidptr,* *int __user *child_tidptr)* *{* *struct task_struct *p;* *int trace = 0;* *long nr;* … *p = copy_process(clone_flags, stack_start, regs, stack_size,* *wake_up_new_task(p, clone_flags);* … *tracehook_report_clone_complete(trace, regs,* *clone_flags, nr, p);*

The '720 Patent	Infringed By
	… *return nr;* *}*
7. A system according to claim 1, wherein the master runtime system process is caused to sleep relative to receiving the process request.	Android includes a master runtime system process that is caused to sleep relative to receiving a process request. *See*  (Dalvik Presentation, Slide 25) Corresponding Dalvik Video at 13:48: "What we do with the zygote, as its name implies, …it sort of sits on a socket and it waits for commands…." *See also*